Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
https://nv.nltu.edu.ua
https://doi.org/10.36930/40290926
Article received 22.10.2019 p.
Article accepted 26.12.2019 p.
UDK 004.[514+4'23+055]

ISSN 1994-7836 (print)
ISSN 2519-2477 (online)

@ Correspondence author
Z. V. Ostapiuk
zoeostapiuk@gmail.com

З. В. Остапюк, Т. О. Коротєєва
Національний університет "Львівська політехніка", м. Львів, Україна

ЗАСТОСУВАННЯ ГРАФІВ ДЛЯ ВІДОБРАЖЕННЯ ЖИТТЄВОГО ЦИКЛУ СУТНОСТЕЙ ПІД ЧАС РОЗРОБЛЕННЯ СИСТЕМИ ОПРАЦЮВАННЯ ВІДГУКІВ БЕЗПОСЕРЕДНІХ КОРИСТУВАЧІВ ПРОГРАМНИХ ПРОДУКТІВ

Важливою для безпосереднього користувача здатністю будь-якого програмного продукту є гнучкість застосування та налаштування. Проблема, описана та частково досліджена у цій науковій роботі, стосується питання забезпечення цієї гнучкості, а саме – підходів до задавання в межах програмної системи набору станів певної сутності, а також накладення обмеження на множину станів, у які згадана вище сутність може перейти, перебуваючи в одному із них. Тут і далі під правилами переходу сутності в різні стани мають на увазі обмеження множини наступних станів. Оглянуто сучасні системи для керування відгуками до програмного забезпечення, як приклад предметної області зі сутностями, які не мають наперед визначеної множини станів та переходів між ними. Проаналізовано основні переваги та недоліки аналогічних систем та їх підходу до зберігання станів. Наведено приклади та описи можливих станів сутностей та правил їх переходів. Досліджено перспективи застосування теорії графів для вирішення поставленої у статті проблеми. На підставі проведеного дослідження спроектовано архітектуру та реалізовано згідно з нею систему, що складається з мобільного та браузерного (веб-сайт та розширення веб-переглядача Google Chrome) клієнтів. Ціль системи – забезпечити проектні команди легким для освоєння засобом для збирання та оброблення різноманітних відгуків безпосередніх користувачів та зацікавлених сторін. Зокрема, розроблена система дає змогу створювати шаблони відгуків із різними наборами полів та різним типом кожного із них. Результати дослідження застосовано для реалізації функціональності зберігання та опрацювання динамічних станів сутності відгуку в межах розробленої програмної системи. Обґрунтовано вибір інтерфейсного рішення для представлення правил переходів між станами сутності для безпосередніх користувачів. Досліджено та застосовано алгоритм перевірки коректності завдання станів та правил їх переходів.

Ключові слова: програмне забезпечення; теорія графів; динамічні стани; алгоритм Тарджана; компоненти сильної зв'язності; орієнтований граф.

Вступ. Щороку кількість та складність програмного забезпечення зростає, дедалі більше областей бізнесу піддаються автоматизації та переведенню в електронний варіант оброблення даних, специфічних для конкретних потреб. Найважливіші сутності з предметних областей стають основою розробки вимог, тестів та коду для створення ПЗ, яким будуть користуватись бізнес-спеціалісти (Evans, 2014).

Враховуючи інтенсивність зростання ІТ сфери у всьому світі, не дивно, що зростає також і важливість якісної підтримки користувачів у їх питаннях до роботи програмних продуктів (Shaw, DeLone & Niederman, 2002). Сьогодні існує безліч систем для отримання та зберігання відгуків безпосередніх користувачів, звітів про дефекти та інших сутностей, пов'язаних із життєвим циклом програмного забезпечення (Desmond, 2017). Основним недоліком їх є складність та незручність у використанні для звичайних користувачів програмного забезпечення, життєвий цикл якого підтримується за допомогою згаданих вище систем (Hryb, 2019). Понад це, зазвичай безпосередні користувачі навіть не мають доступу до цих "внутрішніх" систем.

Відгуки до програмного забезпечення можуть стосуватись будь-якого аспекту розробки системи. Це може бути як звіт про дефект, запит на створення додаткової функціональності, так і запити, які, на перший погляд, не стосуються системи, але тим не менш повинні бути задокументовані та відповідно опрацьовані (Maalej & Nabil, 2015). Сюди належать запити на отримання дозволу для створення, наприклад, нового акаунту в певній системі, повідомлення про оновлення версії ПЗ, що розробляється.

Розроблена система має на меті надати зручний інструмент налаштування складних сутностей, які відрізняються своєю структурою, а також можливою поведінкою в сенсі зміни станів та правил переходу з одного стану в інший. Також система надаватиме можливість створювати шаблони відгуків, які визначатимуть набір атрибутів кожного з них. Така функціональність зробить систему практичною для проектної команди.

Інформація про авторів:
Остапюк Зоя Вікторівна, бакалавр, кафедра програмного забезпечення. **Email:** zoeostapiuk@gmail.com
Коротєєва Тетяна Олександрівна, канд. техн. наук, доцент, кафедра програмного забезпечення.
Email: tetyana.o.korotyeyeva@lpnu.ua
Цитування за ДСТУ: Остапюк З. В., Коротєєва Т. О. Застосування графів для відображення життєвого циклу сутностей під час розроблення системи опрацювання відгуків безпосередніх користувачів програмних продуктів. Науковий вісник НЛТУ України. 2019, т. 29, № 9. С. 147–152.
Citation APA: Ostapiuk, Z. V., & Korotyeyeva, T. O. (2019). Application of graphs for issue lifecycle visualization in the system of software end-user feedbacks management. *Scientific Bulletin of UNFU, 29*(9), 147–152. https://doi.org/10.36930/40290926

Частиною розробленої системи є розширення вебпереглядача Google Chrome, яке дасть змогу безпосереднім користувачам створити відгук, не покидаючи сторінки, на яку цей відгук необхідно зробити. Це зекономить час користувачів та не буде потреби вивчати складні системи, що є основним недоліком великих існуючих рішень (Hryb, 2019).

Аналіз актуальних досліджень і публікацій. Аналіз досліджень показав, що такі програмні засоби, як Jira і Redmine є рекомендованими для використання під час керування складними проектами (Sarkan, Ahmad & Bakar, 2011). Ці системи побудовані навколо концепції проблеми – сутності, яка описує певну зміну, чи проблему, яку необхідно вирішити. Дослідження цих програмних продуктів є важливим, адже вони напряму пов'язані з тематикою статті та розробленої під час дослідження системи.

Найчастіше програмним забезпеченням для керування змінами та відгуками безпосередніх користувачів користуються на етапі підтримки продукту (Bjerknes et al., 1991). Роботи, пов'язані з підтримкою, спричинені діяльністю користувачів. Тому компанії, що розробляють програмне забезпечення, повинні надавати ефективний сервіс, який допоможе користувачам з їх питаннями; вони зацікавлені в якісному та швидкому процесі отримання найновішої інформації від зацікавлених сторін.

Деякі дослідники наголошують на тому, що на таких етапах життєвого циклу ПЗ, як тестування та підтримка майже вся робота розробників спрямована на керування змінами (Mäkäräinen, 2000). До того ж, найдовшим етапом життєвого циклу вважають етап підтримки (Sokappadu, Mattapullut, Paavan & Ramdoo, 2016). Враховуючи довготривалість та вагомість залучення інженерів в такій діяльності, можна зробити висновок, що дослідження процесів цього етапу та створення відповідних програмних систем є актуальним. Цими процесами і є отримання та опрацювання відгуків користувачів із проханням виправити помилки чи розробити нову функціональність.

Розроблення системи для опрацювання даних про побажання/скарги безпосередніх користувачів є виправданою й тому, що інформаційні технології надають можливість спеціалістам навчатись з досвіду минулих проектів (Arain, 2008).

Враховуючи тенденції до автоматизації та машинного оброблення будь-яких даних, зокрема відгуків (Bukhsh, Arachchige & Malik, 2018), об'єми яких не піддаються людському опрацюванню, важливою перевагою системи має стати її можливість налаштовування користувацьких полів у відгуках. Це дасть змогу спеціалістам з машинного навчання отримувати структуровані дані для кожного відгуку/дефекту для подальшої категоризації, оцінки та аналізу загалом.

Детально моделі життєвого циклу дефекту вже описані в багатьох публікаціях, запропоновано безліч варіантів із різними станами та можливими переходами. Наприклад, найпростішу модель життєвого циклу дефекту зображають так (Nindel-Edwards et al., 2006): 1) проблему знайдено; 2) виконується робота над виправленням проблеми; 3) перевірка рішення на коректність; 4) закриття.

Очевидно, цей набір не є достатнім для коректної документації та опису проблеми. Процес роботи над відгуком починається, коли виявляється потреба в його рішенні і закінчується, коли дефект буде усунутий. Ця потреба з'являється в той момент, коли користувач створює запит, який потім буде очікувати, щоб команда проаналізувала, оцінила, надала пріоритет проблемі (Black, 1999). Тут простежується одне із найважливіших правил переходів стану відгуку. Початковим є стан, який зазвичай називають "Новий"/"Створений"; він символізує наявність нової проблеми, з якою ще не працювали. Тому наступним можливим станом є той, який би показував, що проблема аналізується/опрацьовується. Такі стани зазвичай називають "В прогресі", "Аналізується".

Окрім створення нового запиту, відгук може стати актуальним внаслідок перегляду старих, відкладених проблем та появи нових вимог чи умов, за яких згадані вище проблеми отримають високий пріоритет в очах замовника чи безпосереднього користувача (Davis, 2003). Тому після успішного релізу керівництво може переглянути старі закриті чи відкладені дефекти і вирішити перевідкрити їх. Правила переходів у такому випадку виглядають так: "Закритий", "Відкрито повторно", "В прогресі" (умовні назви). Основною ідеєю є те, що має бути можливість перевести сутність із стану, що символізує завершеність роботи ("Закритий"), не просто у "Відкритий", а в такий стан, який би показував, що поточна сутність колись була вирішеною.

Важливим станом є також той, який би вказував на те, що проблема не актуальна, бо її неможливо відтворити (Black, 1999). У такому випадку не варто видаляти відгук. Наявність такого стану допомагає команді зрозуміти та потенційно виправити проблеми, присутність яких визначається певними факторами, які не завжди є відображені. Для збереження історії роботи над сутністю та відстежування змін у багатьох запропонованих життєвих циклах (Nindel-Edwards et al., 2006) у стан "Неможливо відтворити" сутність переходить лише із станів, які символізують активну роботу ("В прогресі"/"Аналіз"/"Обговорення"). Вважають (Black, 1999), що такий висновок можна зробити лише після опрацювання відгуку.

Часто трапляється, що новий відгук вже був створений раніше. Такі випадки особливо є актуальними у проектах, де використовуються автоматичні системи надсилання звітів про збої (McLaughlin, 2004). У деяких випадках дублювання очевидне, як наприклад, коли звіти містять ідентичні тексти з описом. В інших випадках звіт потребує детальнішого огляду і тоді дефект проходить такий приблизний життєвий цикл: "Новий", "Відкритий", "В прогресі", "Дублікат", "Закритий". Тобто правило переходу в такий стан схоже на правило переходу в стан неможливості відтворення.

Популярним рішенням цієї проблематики на ринку є згадана вище система Jira, яка дає змогу створювати відгуки за шаблонами з унікальним набором полів та атрибутів (https://jira.atlassian.com). Перевагами цієї системи є:

- можливість формувати свої типи завдань;
- створення флову, що містить усі дозволені переходи для завдання;
- широка підтримка звітності та побудови графіків;
- JQL (англ. *Jira Query Language*) – потужний засіб для пошуку даних, мова, схожа на SQL, створена для розширеного пошуку, сортування, вибірки.

Система Jira є популярною і для розробників. API Jira використовують багато компаній для інтеграції із власним програмним забезпеченням (Ortu, Destefanis, Kassab & Marchesi, 2015).

Також популярним, але менш функціональним, є рішення Trello (https://trello.com). Перевагами цієї системи є:

- можливість безкоштовного користування;
- підтримка налаштування різних станів сутностей за допомогою створення списків, у межах яких можна переміщати завдання.

Популярним та відомим є рішення Redmine (https://www.redmine.org). Redmine – це гнучка веб-програма для управління проектами. Написана за допомогою фреймворку Ruby on Rails система є кросплатформна і може використовуватись з багатьма базами даних. Redmine є безкоштовним та з відкритим вихідним кодом. Серед переваг варто виділити такі: підтримка декількох проектів; гнучкий контроль доступу на підставі ролі; діаграма та календар Ганта; спеціальні поля для питань, часових записів, проектів та користувачів.

У цій системі реалізовано підтримку користувацьких життєвих циклів сутностей за допомогою квадратної матриці з вказаними раніше користувачем станами. Значення в цій матриці вказують на наявність переходів між станами.

Аналіз систем-аналогів показав, що ПЗ для керування відгуками до інших програмних систем є актуальним та перспективним у сенсі інновацій в підходах до вирішення проблем у цій предметній області. Наприклад, підхід Redmine не є дружнім для простого користувача, він вимагає більше часу на ознайомлення, ніж, наприклад, візуальний підхід, як у системі Jira. Серед недоліків Jira серйозним є складність інтерфейсу та інші недоліки, пов'язані зі складністю налаштування інфраструктури проекту (Hryb, 2019). Недоліком Trello є відсутність будь-яких правил, які б обмежували переходи завдання між списками, а отже, і зміну їх статусу.

Отже, існує безліч можливих шляхів для зображення життєвого циклу запиту (дефект, відгук і т.ін.), проте Голдін і Рошел у своїй лекції "Software Development Bug Tracking: Tool Isn't User Friendly or User Isn't Process Friendly" наголошують, що в підсумку саме гнучкість процесів визначає їх корисність. Це наштовхує на думку, що розроблення системи, яка містила б усі можливі статуси та всі можливі переходи між ними, щоб мати змогу підтримувати будь-яку із відомих моделей життєвого циклу дефекту, не є доцільним. Краще надати можливість, власне, проектній команді обирати, яка модель є найбільш актуальною саме для її проекту.

Об'єкт дослідження – процес отримання та опрацювання відгуків безпосередніх користувачів до програмного продукту.

Предмет дослідження – методи зберігання сутностей, структура яких невідома заздалегідь, а також їх можливих станів та переходів між ними; забезпечення обмеження множини можливих наступних станів для збереження цілісності бізнес-логіки системи, в межах якої фігурує сутність.

Мета дослідження полягає у створенні системи для роботи з відгуками безпосередніх користувачів, яка б продемонструвала розроблений підхід до збереження та опрацювання сутностей із складними та динамічними наборами станів. Серед основних *завдань дослідження* варто виділити:

1. Дослідження та вибір моделі для зберігання станів і переходів між ними, аналіз рішення систем-аналогів проблеми відображення та редагування списку станів і дозволених переходів.
2. Розроблення архітектури системи для отримання, зберігання та опрацьовування відгуків. Проектування оптимального інтерфейсу користувача як для веб-сайту, так і для розширення веб-переглядача.
3. Розроблення та реалізація підходу для зберігання даних про статуси відгуків та можливі переходи між ними в обраному сховищі даних із урахуванням реляційності СУБД, що використовуватиметься і для збереження всіх інших даних системи.
4. Вибір та обґрунтування рішення для зручного відображення усіх можливих станів сутності у зрозумілому вигляді безпосередньому користувачу. Вибір програмного засобу та реалізація редактора для зміни кількості станів, переходів між ними, зміни умовної назви станів та переходів.
5. Дослідження можливих підходів та реалізація перевірки вказаного користувачем графу переходів станів на коректність у межах бізнес-правил системи, а саме: більше ніж один статус; відсутність зациклення; існування єдиного початкового статусу; відсутність ізольованих вершин.

1. Вибір моделі станів сутності та переходів між ними. З аналізу систем-аналогів зроблено висновок, що найбільш зручним та природним відображенням усіх можливих станів сутності та переходів між ними є орієнтований граф.

Як альтернатива, існує можливість відображення переходів за допомогою матриці, де рядки та стовпці є списком станів, а значення на перетині рядка та стовпця позначає, чи можна перейти від стану, що є в рядку, в стан, що є у стовпці.

Нехай n – кількість всіх можливих станів сутності. Тобто, $s_i, i=\overline{1,n}$ – стан в i-му рядку матриці, а $s_j, j=\overline{1,n}$ – стан в j-ому стовпці матриці. Матриця переходів – $M \in \mathbb{R}^{(n\times n)}$. Якщо $M_{i,j}=1$, то існує перехід від стану s_i до стану s_j Оскільки немає сенсу в декількох однакових переходах з вершини s_i в s_j, то в такій матриці суміжності можуть бути лише значення 1 або 0 (існує перехід чи він відсутній).

Оскільки зображення графу сприймається наочніше та швидше, ніж аналіз рядків та стовпців матриці суміжності, обрано саме таку модель для відображення в інтерфейсі користувача.

Схеми переходів станів сутностей доцільно зображати у вигляді *орієнтованого графу*. У такому графі кожна вершина представляє *стан* сутності, а дуга графу є *переходом*, причому сутність не може перейти з одного стану в той самий.

Формально, граф переходу станів $D=(V,E)$, де V – множина всіх станів сутності, а E – множина пар станів, де один з них є попереднім станом, а інший тим, в який сутність переходить (Georgiadis et al., 2014). Тобто дуга між станами v_0 і v_1 є інцидентною до них, v_0 є початковим станом, v_1 – наступним.

Модель обрано із врахуванням накладених обмежень щодо переходу сутності в різні стани, тому граф є *орієнтованим*, що дає змогу реалізувати заборону переходів від будь-якого стану в наступний і назад.

Для того, щоб сутність перейшла з початкового стану в інший v_k, необхідно пройти *маршрут* вигляду $v_0, e_1, v_1, e_2, v_2, ..., e_k, v_k$. Вершини (стани v) в цьому маршруті можуть повторюватись.

Важливим обмеженням є *досяжність* кожної вершини з початкової v_0. Це обмеження дає змогу перевірити, чи введений користувачем граф переходів станів за допомогою графічного редактора не має вершини (стану v_m), в яку сутність ніколи не перейде, через те, що не існує *шляху* від v_0 до v_m.

2. Реалізація архітектури програмного продукту. Для розроблення програмного продукту, який продемонстрував би використання обраної моделі, було створено декілька незалежних проектів для браузерного та мобільного клієнтів, а також для розширення браузерного переглядача Google Chrome. Основна логіка роботи з сутностями була реалізована на серверній частині.

Для збереження версійності було створено репозиторій для коду та інших артефактів усіх проектів та підпроектів рішення, а також для збереження історії змін. Для реалізації рішення було використано такі технології: Android, Kotlin, Firebase, jQuery, Angular, Angular Material, Bootstrap, SignalR, Entity Framework. Під час розроблення частини функціональності, яка забезпечуватиме роботу зі станами сутностей, використовувались:

- Сховище даних MS SQL Server 2017 для збереження та опрацювання даних для подальшого представлення та використання схеми зміни станів, а також для всіх інших сутностей, як користувачі, проекти, файли, коментарі та ін.
- Мова C# для комунікації з базою даних та передачі даних на клієнтську частину. На серверній частині також реалізовано клієнт системи Jira для інтеграції (Ortu, Destefanis, Kassab & Marchesi, 2015), яка дасть змогу імпортувати/експортувати існуючі сутності за допомогою налаштування проекцій.
- Мова TypeScript разом із фреймворком Angular на клієнтській частині, де реалізовано редактор станів та логіку перевірки введених даних для збереження станів та переходів між ними.

3. Збереження моделі станів і переходів між ними у сховищі даних. Для зберігання графу станів та переходів між ними обрано СУБД MS SQL Server 2017, де будуть зберігатись й інші сутності з предметної області керування відгуками до програмних продуктів. А саме граф та інформацію про зв'язки між його вершинами зображено в ER (англ. *Entity Relationship*) моделі (рис. 1), на підставі якої спроектовано відповідні таблиці. Кожен граф застосовується в шаблонах до відгуку, адже відгуки можуть мати різний зміст (різні поля, відповідно й різний життєвий цикл, який у системі представлений графом).

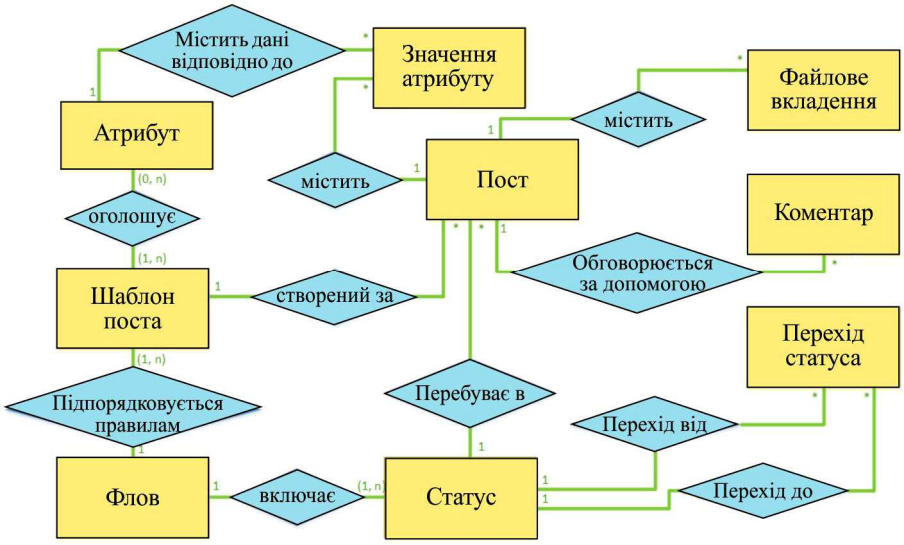

Рис. 1. ER-діаграма відгуку будь-якого типу та його зв'язків

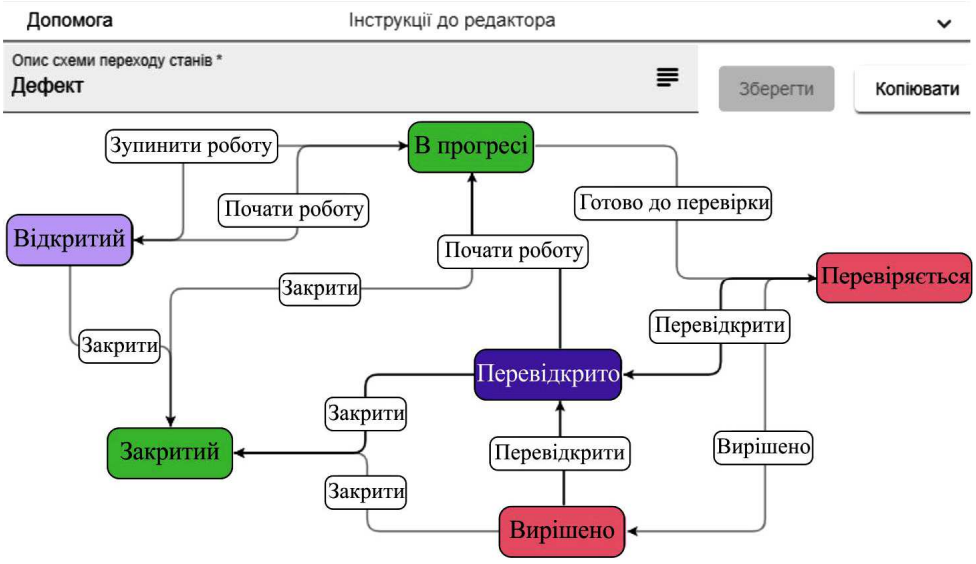

Рис. 2. Приклад графу переходів станів для певного відгуку

На підставі ER-діаграми було створено фізичну діаграму частини бази даних, де будуть зберігатись графи переходів станів.

4. Графічний редактор графу переходів станів. Для того, щоб користувачі мали змогу якомога зрозуміліше та зручніше задавати графи переходу станів, потрібно розробити компонент графічного інтерфейсу користувача, який дав би змогу за допомогою миші інтуїтивно створювати та редагувати схему станів.

Графічний редактор виглядає як інтерактивне полотно (рис. 2), за допомогою якого користувач може додавати, видаляти, перейменовувати стани та переходи між ними. Редактор також дає змогу встановлювати зв'язки між статусами за допомогою перетягування миші від першої вершини до наступної.

Для реалізації графічного редактора обрано бібліотеку go.js, яка застосовується для побудови графіків. За допомогою інструментарію цієї бібліотеки було реалізовано переміщення елементів, створення нових станів, створення зв'язку між станами, а також підсвічення попередніх та можливих наступних станів при наведенні миші на певний стан.

Відбір множин станів для поточної вершини v_c з множини V та колір їх підсвічування повинен відбуватись за такими правилами:

$$V_{prev} = \{v_k \in V \mid \exists\{v_k, v_c\}\}\,;\; V_{next} = \{v_k \in V \mid \exists\{v_c, v_k\}\}\,.$$

Вершини із множини V_{prev} підсвічуються червоним кольором, що означає, що з поточної вершини v_c не можна повернутись в будь-яку вершину V_{prev}. Вершини V_{next} підсвічуються зеленим кольором, що означає, що сутність має можливість перейти в один зі станів цієї множини. Якщо існує зворотній зв'язок між двома вершинами, то перевага надається зеленому кольору.

5. Перевірка графу станів на досяжність кожної вершини з початкової. Для того, щоб визначити, чи існують вершини, недосяжні з початкової, необхідно проаналізувати граф на наявність *компонент зв'язності графу*. Орієнтований підграф є компонентом зв'язності графу, якщо існує *шлях* між всіма парами його вершин. Окрема вершина також може бути компонентом зв'язності графу, наприклад в ациклічному графі, або, коли ця вершина не є частиною підциклу в графі. Для виокремлення компонент зв'язності використано алгоритм Тарджана, який базується на таких фактах (Tarjan, 1979):

1. DFS пошук (depth-first search) є основою для DFS дерева/лісу;
2. Компоненти зв'язності формують піддерева DFS дерева (Tarjan, 1971);
3. Якщо можливо знайти вершину такого піддерева, можна зберегти всі вузли в цьому піддереві і тоді це буде одним компонентом зв'язності.

Вхідними даними алгоритму є вершини орієнтованого графу разом із інформацією про дуги. Результатом роботи алгоритму є погруповані вершини, які і є компонентами зв'язності.

Завданням алгоритму є пошук всіх компонент зв'язності у графі. Це вимагає проходження кожної вершини та повторення кроків для кожного стану. Оскільки для вирішення поставленої задачі, а саме визначення, чи існують недосяжні вершини з початкового стану, необхідно проаналізувати граф відносно початкової вершини, то достатньо лише одного проходження всіх кроків для неї.

Основна ідея алгоритму полягає у пошуку в глибину (Tarjan, 1971) з початкового стану (його обирає користувач). При цьому кожна вершина графу відвідується лише один раз. Поточні вершини зберігаються у стеку в такому ж порядку, в якому вони були відвідані. На відміну від звичайного пошуку в глибину, вершина не видаляється зі стеку, якщо існує шлях від неї до іншої вершини, що є в стеку перед нею. Такий шлях означає циклічну зв'язаність між вершинами, що присутні в стеку (Marczyk, 2008).

Після закінчення усіх рекурсивних відвідувань сусідніх вершин поточної вершини $v_П$ маємо інформацію про те, чи існує шлях від $v_П$ до якоїсь вершини, що була додана у стек раніше. Якщо такий шлях існує, то рекурсія повинна розгортатись далі до попередніх викликів, аж поки не повернеться до вершини, яка не має шляху до будь-якої іншої попередньої. При цьому вершини залишаються у стеку. Це і відрізняє алгоритм від простого пошуку в глибину. Отже, коли виконання алгоритму дійде до вершини $v_Г$, яка є "коренем" щойно знайденого піддерева, необхідно видалити по одній вершині зі стеку, аж до $v_Г$ і це буде черговим компонентом зв'язності.

Продовжуючи ці ж дії для кожного сусіда вершини, що вибрана початковою, будуть знайдені всі компоненти зв'язності графу. Вихідними даними алгоритму є список компонент зв'язності. Щоб перевірити, чи немає у графі вершин, які є недоступними з початкової, необхідно для кожної вершини з тих, що є вказаними у вхідному графі, перевірити, чи існує вона в одній з отриманих компонент зв'язності.

У реалізованому програмному продукті описана вище перевірка здійснюється кожного разу, коли користувач додає чи видаляє вершину чи дугу. Якщо перевірка є неуспішною, то користувач не зможе зберегти граф, з'явиться відповідне повідомлення про помилку.

Висновок. Проаналізовано системи для відстежування та опрацювання відгуків до програмного забезпечення, а саме – сучасні підходи до роботи з численними станами сутностей та правилами їх зміни. На підставі аналізу обрано найдоцільніший підхід до зберігання та оброблення набору станів.

Використано теорію графів для вибору моделі зображення станів та їх переходів, а саме – обрано орієнтований циклічний граф, в якому вершини є станами, а дуги – переходами. Застосовано алгоритм Тарджана, який базується на DFS-пошуку, для перевірки досяжності кожної вершини з початкової.

Розроблено програмну систему, яка складається з мобільного клієнта для створення та перегляду власних відгуків, розширення переглядача Google Chrome для швидкого звітування з можливістю автоматичного отримання останніх скриптових помилок з поточної сторінки та веб-додаток для адміністрування проектів. Спроектовано модель бази даних для збереження відгуків, шаблонів для них, додаткових властивостей відгуків, а також наборів станів та переходів між ними.

Змодельовано та розроблено компоненту інтерфейсу користувача для створення та редагування властивостей графу переходу станів, який включає зручну фун-

кціональність створення та переміщення складових графу за допомогою миші.

Одним із важливих результатів дослідження, що представлено у цій роботі, є реалізований модуль керування графом переходів станів, який можна використати й для іншого програмного забезпечення. Це допоможе під час розроблення майбутніх програмних продуктів, де також буде потреба у динамічному створенні станів, залежно від вимог, які диктує бізнес-логіка.

References

Arain, F. (2008). IT-based approach for effective management of project changes: A change management system (CMS). *Advanced Engineering Informatics*, *22*(4), 457–472. https://doi.org/10.1016/j.aei.2008.05.003

Aspvall, B., Plass, M. F., & Tarjan, R. (1979). A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, *8*(3), 121–123. https://doi.org/10.1016/0020-0190(79)90002-4

Atlassian. (2019). Are you looking for Jira? The #1 software development tool used by agile teams. Retrieved from: https://jira.atlassian.com.

Bjerknes, G., Bratteteig T., & Espeseth T. (1991). Evolution of finished computer systems. The dilemma of enhancement. *Scandinavian Journal of Information Systems*, *3*, 25–45.

Black, R. (1999). *Managing the testing process.* Redmond, Wash.: Microsoft Press.

Bukhsh, F., Arachchige, J., & Malik, F. (2018). Analyzing Excessive user Feedback: A Big Data Challenge. *International Conference on Frontiers of Information Technology* (*FIT*).

Davis, A. (2003). The art of requirements triage. *Computer*, *36*(3), 42-49.

Desmond, C. (2017). Project management tools-software tools. *IEEE Engineering Management Review*, *45*(4), 24–25.

Evans, E. (2014). *Domain-driven design*. Boston, Mass.: Addison-Wesley.

Georgiadis, L., Giuseppe, F., Laura, L., & Parotsidis, N. (2014). *2-edge Connectivity in Directed Graphs*. In Proc. 26th ACM-SIAM Symp. on Discrete Algorithms, 1988–2005.

Hryb, D. (2019). *7 reasons why people say they hate using Jira*. Retrieved from: https://deviniti.com/atlassian/why-people-say-they-hate-using-jira/.

Maalej, W., & Nabil, H. (2015). Bug report, feature request, or simply praise? On automatically classifying app reviews. *IEEE 23rd International Requirements Engineering Conference* (*RE*).

Mäkäräinen, M. (2000). *Software change management processes in the development of embedded software:* Dissertation. Espoo: VTT Technical Research Centre of Finland, 185 p.

Marczyk, A. (2008). *Cycles in graphs and related problems. Dissertationes Mathematicae – DISS MATH*, 1–98.

McLaughlin, L. (2004). In the news – Automated bug tracking: the promise and the pitfalls. *IEEE Software*, *21*(1), 100–02.

Nindel-Edwards, Jim, & Steinke, Gerhard. (2006). A Full Life Cycle Defect Process Model That Supports Defect Tracking, Software Product Cycles, And Test Iterations. *Communications of the IIMA*, *6*(1), 16.

Ortu, M., Destefanis, G., Kassab, M., & Marchesi, M. (2015). Measuring and Understanding the Effectiveness of JIRA Developers Communities. *IEEE/ACM 6th International Workshop on Emerging Trends in Software Metrics.*

Redmine. (2019). Redmine is a flexible project management web application. Retrieved from: https://www.redmine.org.

Sarkan, H., Ahmad, T., & Bakar, A. (2011). Using JIRA and Redmine in requirement development for agile methodology. *Malaysian Conference in Software Engineering.*

Shaw, N., DeLone, W., & Niederman, F. (2002). Sources of dissatisfaction in end-user support. *ACM SIGMIS Database*, *33*(2), 41.

Sokappadu, B., Mattapullut, S., Paavan, R., & Ramdoo, V. (2016). Review of Software Maintenance Problems and Proposed Solutions in IT consulting firms in Mauritius. *International Journal of Computer Applications*, *156*(4), 12–20.

Tarjan, R. (1971). *Depth-First Search and Linear Graph Algorithms*. Foundations of Computer Science, IEEE Annual Symposium on, 114–121.

Trello. (2019). Trello sposobstvuet bolee tesnomu sotrudnichestvu i uvelicheniiu effektivnosti raboty. Retrieved from: https://trello.com. [In Russian].

Z. V. Ostapiuk, T. O. Korotyeyeva

Lviv Polytechnic National University, Lviv, Ukraine

APPLICATION OF GRAPHS FOR ISSUE LIFECYCLE VISUALIZATION IN THE SYSTEM OF SOFTWARE END-USER FEEDBACKS MANAGEMENT

Software flexibility has always been one of the most important features for all end-users. It includes the ease of infrastructure configuration and reaches possibilities for software functionality settings. In this paper, a problem of reaching the flexibility desired by end-users is described and partially investigated, as regards approaches to entity configuration. This configuration includes setting up functionality to allow end-users configure possible entity states that cannot be known at the stage of the concrete software development. Hereafter, we refer to constraining the collection of next reachable states of some entity as transition rule. Existing software feedback management software products are reviewed that serve as an example of a domain with entities that have complex and dynamic sets of states, and do not have predefined collections of them as well as rules for their transition. The main advantages and disadvantages of similar systems and their approach to entities' state management are analyzed. Examples and descriptions of possible states of entities and rules of their transitions are given. The prospects of applying the graph theory to solve the problem posed in the article are investigated. Based on the research, the architecture was designed. A software system was implemented according to the architecture; it consists of mobile and browser (both as a web-site and a Google Chrome browser extension) clients. The goal of the system is to provide project teams with an easy-to-use tool for collecting and storing different feedbacks from end-users and stakeholders. In particular, the system designed and developed during the investigation provides the ability to create feedback templates with different sets of fields and different types of each. The article focuses on the process of managing feedback states and the implementation details of this functionality in the system that was developed. The results of the study conducted in the article were applied to store and validate the dynamic states of the feedbacks to software products. The choice of an interface solution to represent the rules of transition between states is justified. The algorithm for checking the correctness of the states and the rules of their transitions in the system for feedback management is investigated and applied. In particular, the application of Tarjan algorithm for validating states and theirs transitions is described as well as implemented in the system that was developed.

Keywords: software; graph theory; dynamic states; Tarjan algorithm; strongly connected components; oriented graph.